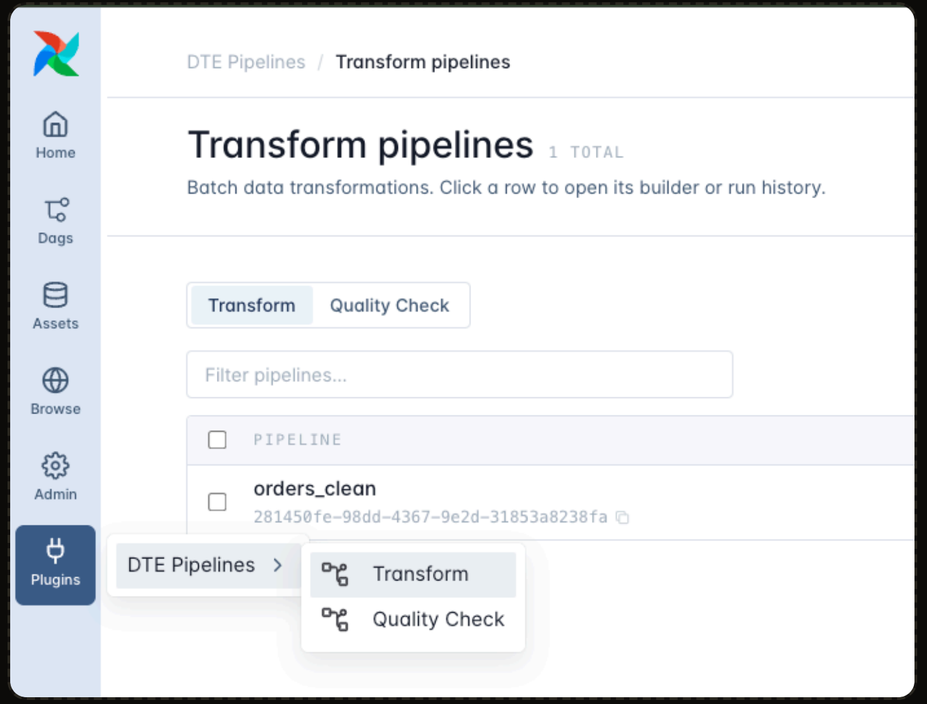
AIRFLOW SUMMIT RECONNECT

Plugin-Maxxing

How we built a *data product* inside Airflow 3

BHAVANI RAVI · FLOWEXPERTS.AI



Bhavani Ravi

FLOWEXPERTS.AI · WE HELP COMPANIES ADOPT OPEN SOURCE TOOLS

2019
Airflow user, from 1.10

2024
Apache Airflow Champion

2025
Speaker at Airflow Summit

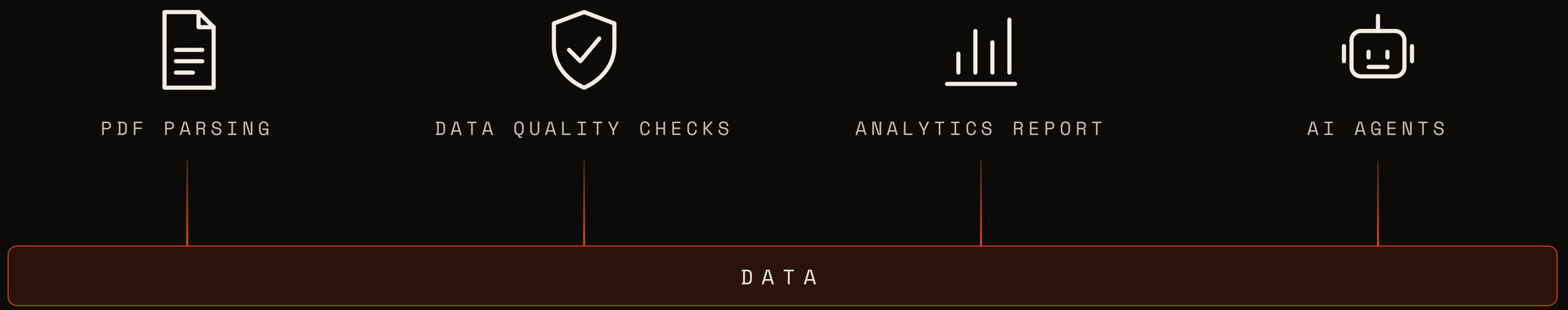
2026
Beyond the DAG hackathon

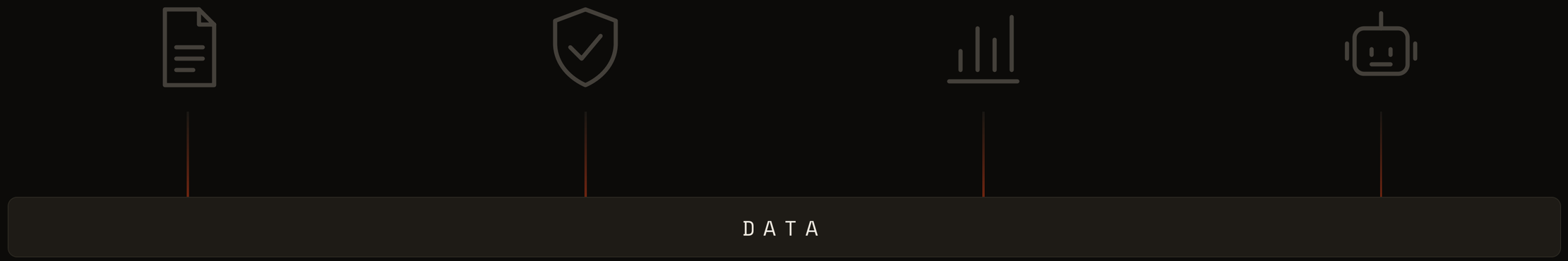
In this *talk*

- ✓ Why I built DTE
- ✓ DTE Demo
- ✓ Build vs Adopt
- ✓ Airflow Plugin 101
- ✓ The DAG factory
- ✓ Gotchas

All problems are *data problems*

Right data, right place → *build anything*





What are these *data problems?*

THE BUCKETS

Extract

Check

Transform

MAP · PIVOT

Load

Present

Use case *vs functions*

USE CASE	EXTRACT	CHECK	TRANSFORM	LOAD	PRESENT
PDF parsing	✓	✓		✓	
Data quality checks		✓			✓
Analytics report			✓	✓	✓
AI agents	✓	✓	✓	✓	

```python
# transform_orders.py
def transform_orders(df):
    df = df.rename(columns={"ord_id": "order_id",
        "cust_id": "customer_id", "amt": "amount",
        "ord_dt": "order_date", "curr": "currency"})
    df["order_id"] = df["order_id"].astype("int64")
    df["customer_id"] = df["customer_id"].astype("int64")
    df["amount"] = df["amount"].astype("float64").round(2)
    df["order_date"] = pd.to_datetime(df["order_date"])
    df["currency"] = df["currency"].str.upper()
    df["order_month"] = df["order_date"].dt.to_period("M")
    df["is_refund"] = df["amount"] < 0
    df = df[df["status"] != "test"]
    return df.drop(columns=["legacy_flag", "tmp_col"])

# transform_customers.py
def transform_customers(df):
    df = df.rename(columns={"cust_id": "customer_id",
        "fname": "first_name", "lname": "last_name",
        "eml": "email", "ph": "phone"})
    df["customer_id"] = df["customer_id"].astype("int64")
    df["email"] = df["email"].str.lower().str.strip()
    df["full_name"] = df["first_name"] + " " + df["last_name"]
    df["phone"] = df["phone"].str.replace(r"\D", "")
    return df.drop(columns=["first_name", "last_name"])
```

```python
# transform_invoices.py
def transform_invoices(df):
    df = df.rename(columns={"inv_no": "invoice_id",
        "inv_dt": "invoice_date", "due": "due_date",
        "tot": "total", "tx": "tax"})
    df["invoice_date"] = pd.to_datetime(df["invoice_date"])
    df["due_date"] = pd.to_datetime(df["due_date"])
    df["total"] = df["total"].astype("float64")
    df["tax"] = df["tax"].fillna(0).astype("float64")
    df["net"] = df["total"] - df["tax"]
    df["days_to_due"] = (df["due_date"]
        - df["invoice_date"]).dt.days
    df["overdue"] = df["due_date"] < pd.Timestamp.now()
    return df.drop(columns=["raw_pdf_path"])

# transform_products.py
def transform_products(df):
    df = df.rename(columns={"sku_cd": "sku",
        "prod_nm": "product_name", "cat": "category",
        "prc": "price"})
    df["sku"] = df["sku"].str.upper().str.strip()
    df["price"] = df["price"].astype("float64").round(2)
    df["category"] = df["category"].fillna("other")
    df["is_active"] = df["status"] == "A"
    return df.drop(columns=["status", "old_sku"])
```

```python
# transform_payments.py
def transform_payments(df):
    df = df.rename(columns={"pay_id": "payment_id",
        "ord_id": "order_id", "mthd": "method",
        "amt": "amount", "pd_at": "paid_at"})
    df["order_id"] = df["order_id"].astype("int64")
    df["amount"] = df["amount"].astype("float64")
    df["paid_at"] = pd.to_datetime(df["paid_at"], utc=True)
    df["method"] = df["method"].map(METHOD_MAP)
    df["is_card"] = df["method"].isin(["visa", "mc"])
    return df.drop(columns=["gateway_raw"])

# transform_shipments.py
def transform_shipments(df):
    df = df.rename(columns={"shp_id": "shipment_id",
        "ord_id": "order_id", "carr": "carrier",
        "shp_dt": "shipped_at", "dlv_dt": "delivered_at"})
    df["shipped_at"] = pd.to_datetime(df["shipped_at"])
    df["delivered_at"] = pd.to_datetime(df["delivered_at"])
    df["transit_days"] = (df["delivered_at"]
        - df["shipped_at"]).dt.days
    df["carrier"] = df["carrier"].str.title()
    df["late"] = df["transit_days"] > SLA_DAYS
    return df.drop(columns=["label_blob"])
```

TRANSFORM_ORDERS.PY

ADDED REMOVED

```
def transform_orders(df):
    df = df.rename(columns={"ord_id": "order_id",
        "cust_id": "customer_id", "amt": "amount",
        "ord_dt": "order_date", "curr": "currency"})
        "ord_dt": "order_date", "curr": "currency",
        "disc": "discount", "chnl": "channel", "rgn": "region"})
    df["order_id"] = df["order_id"].astype("int64")
    df["customer_id"] = df["customer_id"].astype("int64")
    df["amount"] = df["amount"].astype("float64").round(2)
    df["amount"] = df["amount"].astype("float64").round(4)
    df["order_date"] = pd.to_datetime(df["order_date"])
    df["discount"] = df["discount"].fillna(0).astype("float64")
    df["channel"] = df["channel"].str.lower().fillna("unknown")
    df["region"] = df["region"].map(REGION_MAP)
    df["net_amount"] = df["amount"] - df["discount"]
    df["currency"] = df["currency"].str.upper()
    df["order_month"] = df["order_date"].dt.to_period("M")
    df["is_refund"] = df["amount"] < 0
    df = df[df["status"] != "test"]
    df = df[~df["status"].isin(["test", "sandbox"])]
    return df.drop(columns=["legacy_flag", "tmp_col"])
    return df.drop(columns=["legacy_flag", "tmp_col", "rgn_old"])
```

Enterprise code reviews are *also a pipeline*

DAG processing + DagBundles made deploys easier. Reviews didn't get faster.



3 hours

5 teams

DEMO

Introducing *DTE*

Data Transformation Engine

DTE.FLOWEXPERTS.AI

What does a platform like DTE *need?*

Extract	Data source · **extraction engine** · Destination
Check	Data source · **check engine**
Transform	Source · **mapping engine** · Destination
Load	Source · Destination
Present	Load · **BI engine**

I build the engines. *Airflow already supports the rest.*

MINE

- Extraction
- Checks
- Mapping
- BI engine

AIRFLOW

- Source
- Destination
- Schedule
- Run history
- Updating rules
- Trigger params
- Retries
- Human in the loop
- Notifications

The bells and whistles are *Airflow features*

VERB	NEEDED	AIRFLOW ALREADY HAS
Connect	Source, Destination	Connections
Schedule	Schedule	Timetables, Assets
Trigger	Trigger params	Manual DagRun + `conf`, REST API
Track	Run history	DagRun / TaskInstance states, grid
Update	Updating rules	DagVersion
Observe	Notifications	Task logs, metrics, notifiers
Fix	Retries, Human in the loop	Retries, clear and re-run, backfill, HITL operators

DTE as an *Airflow plugin*

How DTE *plugs into Airflow*

PIP INSTALL DTE

PLUGIN	DtePlugin · fastapi_apps + react_apps	→	API-SERVER API + pages in Airflow's nav
DB	dte schema · own Alembic	→	METADATA POSTGRES specs + revisions next to Airflow's tables
DAG FACTORY	dags/dte_dags.py → build_dags()	→	DAG-PROCESSOR one DAG per pipeline, run by DtePipelineOperator

AIRFLOW 3



BORROWED FROM AIRFLOW

Auth + RBAC
auth manager

Credentials
Connections

Scheduling
cron timetables

Recovery
clear, re-run, backfill

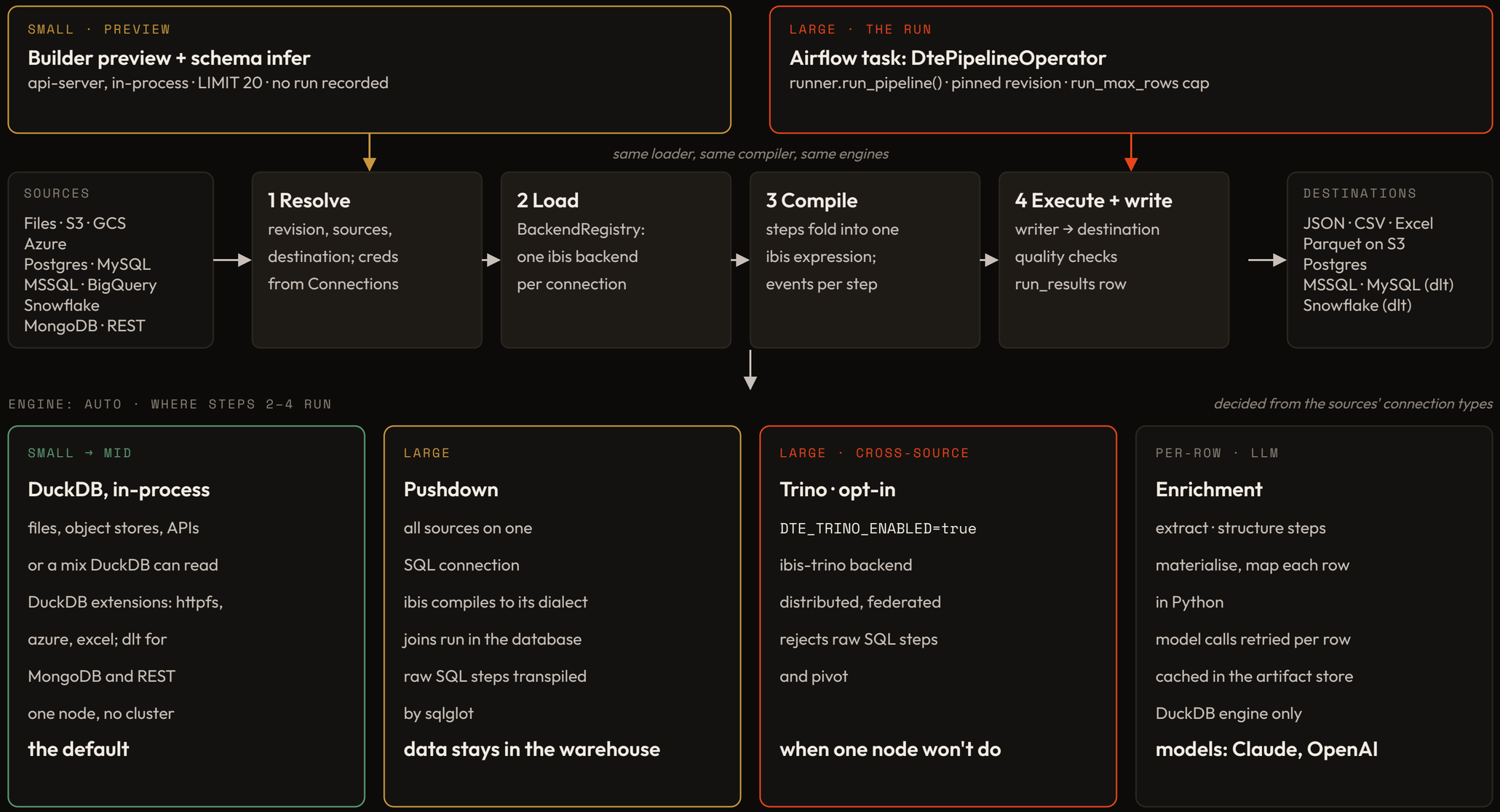
Versioning
DagVersion per save

Observability
task logs, run history

UI shell
rail, theme, light/dark

Deploy
one plugin entry point

DTE · HOW A PIPELINE RUNS



DTE *Package*

```
backend/
├─ pyproject.toml      # entry point: airflow.plugins
└─ dte/
   ├─ airflow/
   │  ├─ plugin.py       # AirflowPlugin: API + nav pages
   │  ├─ dag_factory.py  # rows → DAGs
   │  ├─ operators.py    # DtePipelineOperator
   │  └─ runner.py · connections.py · auth.py
   └─ api/               # the FastAPI app
frontend/ → dte.umd.js   # built React bundle
```

PLUGIN 101 · PYPROJECT.TOML

Register it: *one entry point*

```
# backend/pyproject.toml
[project.entry-points."airflow.plugins"]
dte = "dte.airflow.plugin:DtePlugin"
```

`pip install` is the deploy.

Subclass *AirflowPlugin*

```python
# dte/airflow/plugin.py
class DtePlugin(AirflowPlugin):
    name = "dte"
    fastapi_apps = [{"app": app, "url_prefix": "/dte", "name": "DTE"}]
    react_apps = [{"name": "Transform", "bundle_url": "/dte/ui/dte.umd.js",
                   "destination": "nav", "category": "DTE Pipelines",
                   "url_route": "dte"}]
```

↑ FASTAPI_APPS: YOUR API ↑ REACT_APPS: YOUR UI

DAG FACTORY

A DAG file that *builds DAGs*

```python
# dags/dte_dags.py
from dte.airflow.dag_factory import build_dags
globals().update(build_dags())
```

```python
# dte/airflow/dag_factory.py (trimmed)
def build_dags() -> dict:
    with Session(sync_engine()) as s:
        latest = s.exec(select(R).distinct(R.pipeline_id)
                        .order_by(R.pipeline_id, R.changed_at.desc())).all()
    dags = {}
    for rev in latest:
        with DAG(dag_id=dag_id_for(rev.pipeline_id), schedule=cron,
                 tags=["dte", revision_tag(rev.id)]) as dag:
            DtePipelineOperator(task_id="run", pipeline_id=rev.pipeline_id,
                                revision_id=rev.id)
        dags[dag.dag_id] = dag
    return dags
```

THE OPERATOR

One operator *runs a pipeline*

```python
# dte/airflow/operators.py (trimmed)
class DtePipelineOperator(BaseOperator):
    def __init__(self, *, pipeline_id, revision_id=None, **kw): # ...

    def execute(self, context):
        conf = context["dag_run"].conf or {}
        revision_id = conf.get("revision_id") or self.revision_id
        record_result(engine, run_id, status="running", ...)
        rev = s.get(PipelineRevision, revision_id)
        conn, creds = to_dte(src.connection_id)   # Airflow Connection
        res = run_pipeline(pipeline, sources, destination, ...)
        record_result(engine, run_id, status=res.status, ...)
        if res.status == "failed_quality":
            raise AirflowFailException("quality checks failed")
```

↑ PINNED REVISION ↑ CREDENTIALS BY CONN_ID ↑ RESULT KEYED BY RUN_ID

REACT_APPS

Native UI, *not an iframe*

```
// vite.config.ts
lib: { entry: "src/plugin.tsx",
       name: "AirflowPlugin",
       formats: ["umd"] },
rollupOptions: {
  external: ["react", "react-dom",
             "react-router-dom"],
  output: { globals: { react: "React", ... } }
}
```

- ✓ Airflow loads the bundle into its own page
- ✓ Airflow hands over its own React, ReactDOM and router
- ✓ Same theme, fonts and router as the rest of Airflow

REACT_APPS · THE BUILD

From React to *dte.umd.js*

```
# frontend/
pnpm install --frozen-lockfile
pnpm build                    # → dist/dte.umd.js

# Dockerfile.airflow: ship it with the API
COPY --from=ui /app/dist /opt/dte/backend/ui
```

FASTAPI_APPS · GOTCHA

Custom *Permissions*

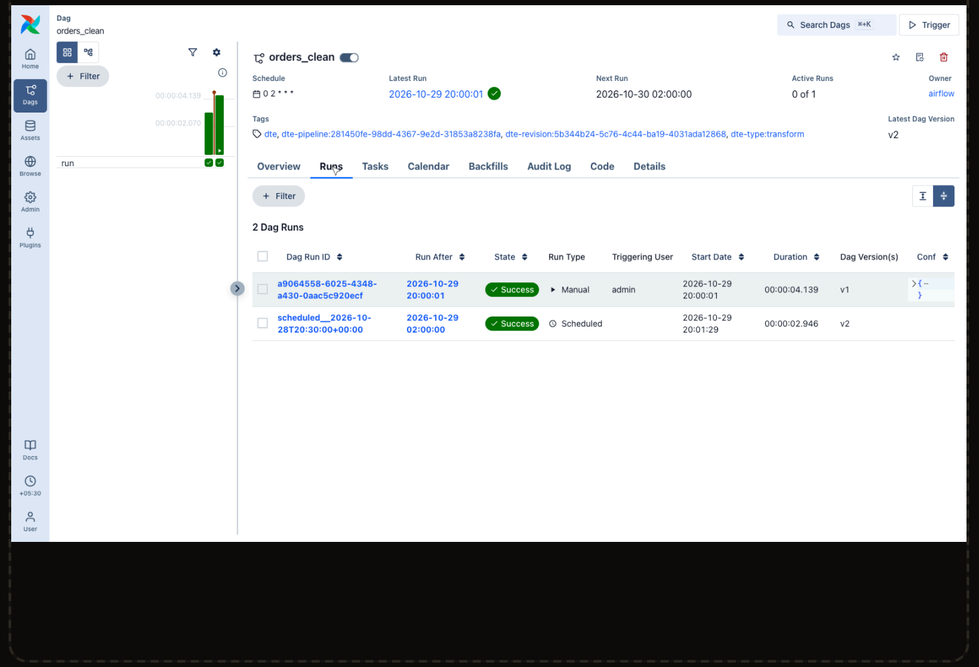
AUTH MANAGER	YOUR CUSTOM PERMISSION CHECK
Simple (the default)	Ignored. Any viewer passes.
FAB	Honoured, if an admin grants it in a role.

DTE asks `is_authorized_dag` instead: read → GET, write → PUT, delete → DELETE, trigger → POST.

Every save is a *new DagVersion*

`save` → `revision` → `dte-revision:<id>` → `DagVersion`

- ✓ The operator runs the pinned revision, never the live row
- ✓ Airflow doesn't serialize custom operator attributes, so the revision rides in a DAG tag

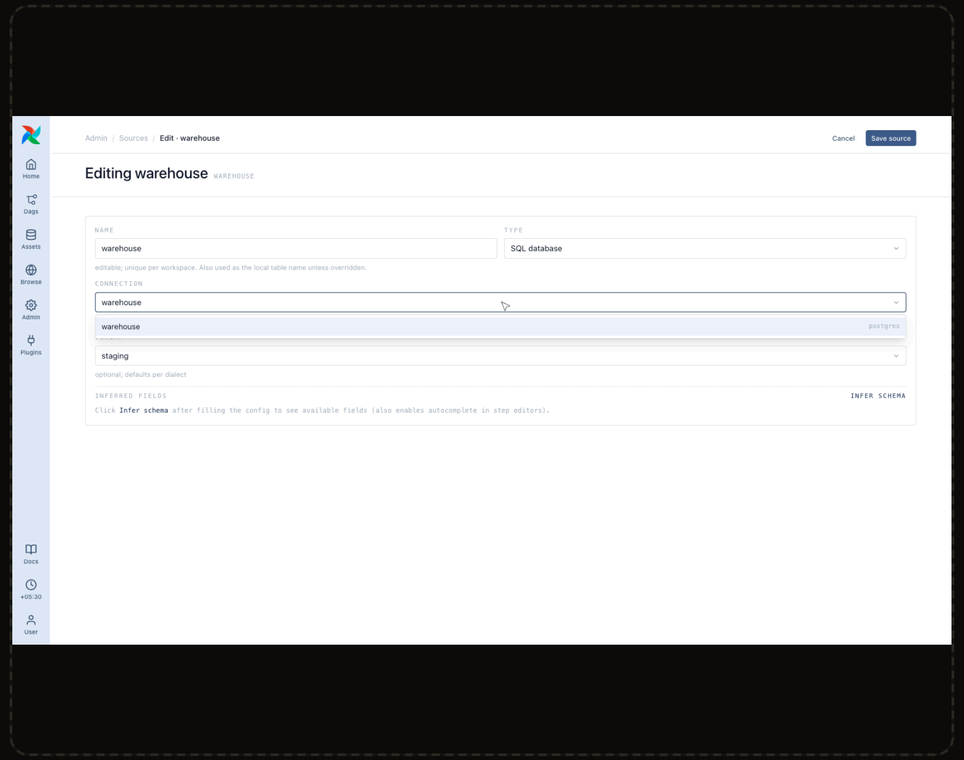


Secret *Management*

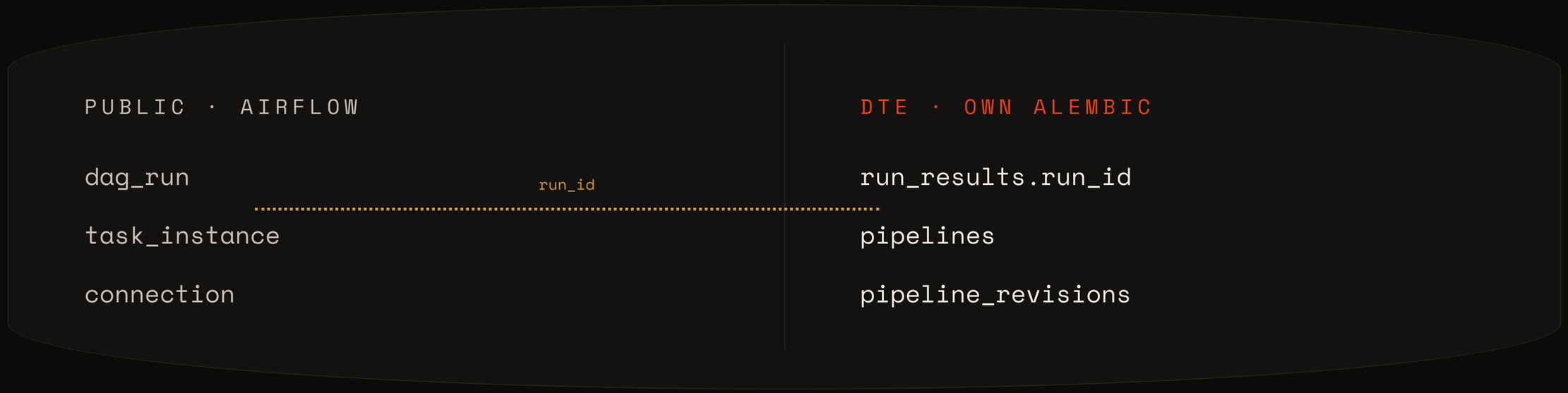
AIRFLOW CONN TYPE	DTE SOURCE
postgres · mysql · mssql · snowflake · mongo	SQL family
aws · wasb · google_cloud_platform	S3 · Azure · GCS / BigQuery
http · fs	REST API · local files

No custom conn type, no hook wrapping. GCP ambiguity settled by one `extra` field.

Gotcha: DTE only sees connections in Airflow's DB. Ones set via `AIRFLOW_CONN_*` env vars or a secrets backend won't show up.



Sharing *Run-ID*



DTE stores specs, revisions, results. Airflow stores status and timings.

BUILD YOUR OWN

Your data product as a plugin, *in five steps*

01

Register

`airflow.plugins` entry point

02

Mount

`fastapi_apps` + `react_apps`

03

Factory

a DAG file that builds DAGs

04

Operator

one thin task, pinned revision

05

Ship

`pip install`

KNOW THE LIMITS

When a plugin is *not enough*

- ✓ Past a few hundred specs, parsing from the DB gets slow. Write DAG files on save instead.
- ✓ Connections set via env vars or a secrets backend don't show up in your UI.
- ✓ You are coupled to an Airflow version. Pin it and test upgrades.
- ✓ Plugins can't add permissions. The simple auth manager ignores custom checks, so map onto DAG access.

Think *Airflow First*

Everything else, you don't build.

YOURS

- UI
- Engine

AIRFLOW

- Connections
- Timetables
- Assets
- DagRun + `conf`
- DagVersion
- Grid + task logs
- Retries
- HITL operators
- Notifiers
- Auth manager
- `react_apps`
- `fastapi_apps`

DTE × Airflow = *SME* 🤝 *Data Engineers*

PDF parsing

New PDFs land in the bucket → one task per document → a human approves low-confidence fields.

`Assets` `Task mapping` `HITL`

Data quality

Hourly checks on every table → a failed check pings the data owner.

`Timetable` `Task mapping` `Notifiers`

Analytics report

Monday 9am refresh → approved before it reaches execs → you know which query version ran.

`Timetable` `HITL` `DagVersion`

AI agents

New docs or tickets refresh the agent's feed → you approve what it may see → stale feeds alert.

`Assets` `HITL` `Notifiers`

Schema mapping

The SME edits the mapping → every save is a new revision → nightly load, failures ping the owner.

`DagVersion` `Timetable` `Connections`

OPEN SOURCE · DTE + AIRFLOW 3

Try *DTE Today*

→ dte.flowexperts.ai

01
Star it

02
Try it

03
Break it